MVC Architecture from Maintenance Quality Attributes Perspective

Safia Nahhas *nahhas.safia@gmail.com*
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah 21577, Saudi Arabia

Abstract

This paper provides an explanatory study on MVC (Model-View-Controller) architecture from the perspective of maintenance. It aims to answer a knowledge question about how MVC architecture supports the maintainability quality attributes. This knowledge boosts the potential of utilizing the maintainability of MVC from several sides. To fulfill this purpose, we investigate the main mechanism of MVC with focusing on maintainability quality attributes. Accordingly, we form and discuss MMERFT maintainability set that consists of Modifiability, Modularity, Extensibility, Reusability, Flexibility, and Testability. Besides investigating the mechanism of MVC regarding MMERFT quality attributes, we explain how MVC supports maintainability by examining measures and approaches such as: complexity of code by using a cyclomatic approach, re-engineering process, use of components, time needed to detect bugs, number of code lines, parallel maintenance, automation, massive assignment, and others. Therefore, this paper is dedicated to providing a concrete view of how MVC gets along with maintainability aspects in general and its several attributes particularly. This view helps to maximize the opportunity of taking advantage of MVC's maintainability features that can encourage reconsidering the maintenance decisions and the corresponding estimated cost. The study focuses on maintainability since software that has high maintainability will have the opportunity to evolve, and consequently, it will have a longer life. Our study shows that MVC generally supports maintainability and its attributes, and it is a recommended choice when maintenance is a priority.

Keywords: MVC Architecture, Maintainability, Modifiability, Modularity, Extensibility, Reusability, Flexibility, Testability.

1. INTRODUCTION

The MVC architecture stands for Model, View, and Controller. It separates business logic from presentation functionality (Majeed & Rauf, 2018). MVC provides a kind of an archetype architecture that helps to build applications with several maintainability features (Kazman, Bianco, Ivers, & Klein, 2020). Studies such as (Xu & Liao, 2021), (Gining et al., 2018) and (Akbar & Handriani, 2018) are examples of utilizing MVC’s maintainability features. We took benefit from the ISO standard's maintainability attributes that are mentioned in (Bass, Clements, & Kazman, 2013) to constitute the MMERFT maintainability set. The MMERFT maintainability set consists of Modifiability, Modularity, Extensibility, Reusability, Flexibility, and Testability. Accordingly, this paper explores many features of MVC’s maintainability through investigating these maintainability quality attributes. In addition to studying most of the maintainability quality attributes that are mentioned in the ISO standard, we investigated the flexibility and extensibility quality attributes with MVC. We found it worthwhile to inspect how MVC supports flexibility, since flexibility is related to adaptive maintenance, as the authors in (Kaur, Kaur, & Kaushal, 2020) pointed out "changes in operation to meet the dynamic needs of any software and adds more flexibility in operation, comes under adaptive maintenance". Also, extensibility is one of the most important advantages of the MVC architecture that facilitates appending new features to the applications and supports adaptive maintenance as well as perfective maintenance (Shekhar, 1998).

The paper provides explanations on how MVC supports maintainability by: 1) investigating the main mechanism of MVC while focusing on MMERFT quality attributes; 2) explaining how MVC supports maintainability by examining measures and approaches such as complexity of code by using a cyclomatic approach, examining how a re-engineering approach can benefit from MVC separated parts, time needed to detect bugs, number of code lines, parallel maintenance, automation, massive assignment, and others; 3) finding and indicating studies and frameworks that touched on maintainability attributes in MVC.

Maintainability is an essential nonfunctional requirement of MVC architecture. Studying MVC architecture with a focus on maintenance matters maximizes the chance of taking advantage of MVC's maintainability. Thus, evolving and lengthening the applications' life. For example, this paper touches on how the MVC architecture can meet the need for code isolation that re-engineering maintenance requires by the separation of concerns that MVC offers. Also, by clarifying that the developer can replace the whole current frontend framework with a totally different frontend framework without any impact on the 'Model part', we can show the high modifiability of MVC. Such examples highlight to which extent MVC architecture facilitates maintainability. Therefore, that can encourage reconsidering the maintenance decisions and the corresponding estimated cost. As the authors indicated in (Kumar, Kumar, & Grover, 2007) and (Molnar & Motogna, 2020), maintenance is one of the major and growing cost concerns.

The rest of the paper is organized as follows: Section 2 presents the methodology of the research. Section 3 examines the MVC architecture and the main maintainability issues. In Section 4, we introduce the MMERFT maintainability set and its quality attributes with MVC. While in Section 5 the discussion of the paper is presented, Section 6 provides the related work. Finally, in Section 7, the paper is concluded.

2. METHODOLOGY

The main aim of this paper is to provide an explanatory study on MVC architecture from the perspective of maintenance, and to answer a knowledge question about how MVC architecture supports the maintainability quality attributes. To achieve this goal, we used the abductive inference and analogic inference designs as illustrated in (Wieringa, 2014). Using abductive inference helped to provide better explanations. We especially focused on architectural explanations. Thus, we analyzed the architectural structure of MVC to understand and analyze the mechanism and capabilities of its components from the maintenance's perspective. To do so, we selected a case study from a population of interest that includes frameworks which adopt the MVC architecture. Accordingly, we selected the Yii framework, which follows the MVC architecture and is used in many real-world applications. The same research could be repeated with other frameworks that follow the MVC architecture. The measures and concepts that we used are based on well-known conceptual framework in software maintenance, such as lines of code, complexity of code, and so on. Data sources included the official Yii website and books such as (Safronov & Winesett, 2014; Winesett, 2012), real Yii applications' source code, project reports and a software engineer. We conducted inductive research since we started by studying and analyzing a case, focusing on its architecture and components, and then came to the conclusion that the MVC architecture supports maintainability with providing explanations.

3. MVC AND THE MAIN MAINTAINABILITY ISSUES

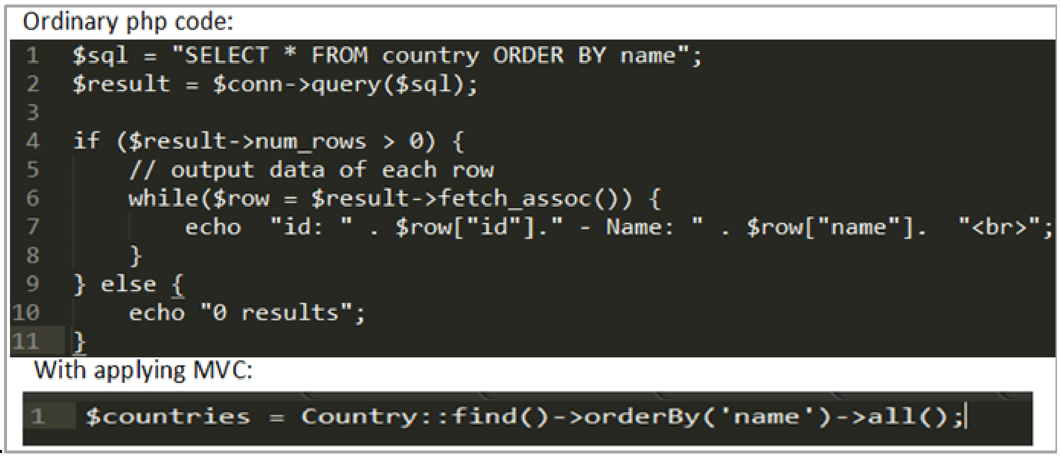
The following issues present the MVC architecture from the perspective of the maintenance.

Re-engineering: Re-engineering facilitates maintaining the software to be adapted to the required changes without affecting its functionality by going through three phases: 1) Reverse engineering, 2) Re-structuring, 3) Forward engineering (Feiler, 1993). Since the reverse engineering as indicated in (Rosenberg & Hyatt, 1996) starts by extracting the requirements and design from the source code, thus the isolation that MVC makes by separating the code parts can facilitate applying this process. Therefore, the reverse engineering process which includes 'slicing or code isolation' can significantly take benefit from the MVC's separated parts.

Lines of codes: one of the issues that affect maintainability is the lines of code (LOC) (Hayes, Patel, & Zhao, 2004). As the authors in (H. Zhang, 2009) pointed out, “Larger modules tend to have more defects”. When the number of code lines is less, maintenance will be easier and the chances of error-prone will be less. Figure 1 shows comparing between two pieces of code that do the same thing, which is selecting all rows from the country table. One of the pieces is ordinary PHP code and the other applies MVC by using the Yii PHP framework. We can notice that adopting MVC reduces the number of lines.

Use of components: Using components helps to decrease the cost of maintenance (Algestam, Offesson, & Lundberg, 2002) and gain efficient maintainability (Sharma, Kumar, & Grover, 2007). One of the basic features of MVC is that it supports reusing components. Reusing as indicated in (Seth, Sharma, & Seth, 2009) improves maintainability. The second piece of code (MVC piece) in figure 1 extends the popular model component ‘Active Record’, which is a basic model component in most MVC applications that reduces the number of lines and improves maintainability.

Complexity of code: The complexity of code is one of the issues that influences maintainability. By using the cyclomatic method, we can compare the complexity of code with following the MVC and without following it. Figure 2 shows the flow graph that is used to compute the complexity of the first piece of code in Figure 1 (code without MVC) as follows: The complexity will be the number of predicate nodes (decision nodes) +1=3. Or it can be computed as the number of edges – number of nodes +2, which will be 8-7+2=3. While the complexity in the second piece (with MVC) is 1. Therefore, we can notice that MVC reduces code complexity, which has a positive impact on maintainability.


Ordinary php code:

```
$sql = "SELECT * FROM country ORDER BY name";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo  "id: " . $row["id"]." - Name: " . $row["name"].  "<br>";
    }
} else {
    echo "0 results";
}
```

With applying MVC:

```
$countries = Country::find()->orderBy('name')->all();
```

FIGURE 1: Comparing the number of lines between the ordinary PHP code and the MVC code.

Abstraction: A high level of abstraction is one of the MVC's characteristics that boosts maintenance (Morales-Chaparro, Linaje, Preciado, & Sánchez-Figueroa, 2007) and (Gamma, Helm, Johnson, & Vlissides, 1993). As we said, MVC supports reusing components, which is an important step towards providing abstraction. Abstraction releases developers from involving them in rewriting code details and making mistakes. For example, in the Yii framework, due to the MVC separation mechanism, by calling ‘findModel($id)’ built-in function, we will have all the model’s data, such as student data, employee data, and so on. Also, ‘hasOne(), hasMany()’ built-in functions could be used to create the joining relationships rather than go into ‘joining issues’ details, even though developers could customize these details.

The time needed to detect bugs: as the code gets smaller, as it will be easier to detect errors. As it is shown in figure 1, MVC design was able to reduce the amount of code, which can help to reduce the time needed to detect errors.

Comments providing: Most popular MVC frameworks offer clear comments on their default behavior. This act helps in providing essential comments to clarify the default behavior of various code parts, which can contribute to supporting the maintenance process. In (Chen & Huang, 2009) the authors show that lacking source code comments highly impacts maintainability. Based on their studies, lacking source code comments is the top factor among 10 higher-severity software development problems which affect software maintainability.

Parallel maintenance: MVC, with its separation parts, supports parallelism in team work that includes parallelism in maintenance work. Authors in (Majeed & Rauf, 2018) stated that "MVC patterns provide the facility of parallel development". Similarly, authors in (Rahmati & Tanhaei, 2021) indicated that by using MVC, many changes were made in parallel by multiple teams.

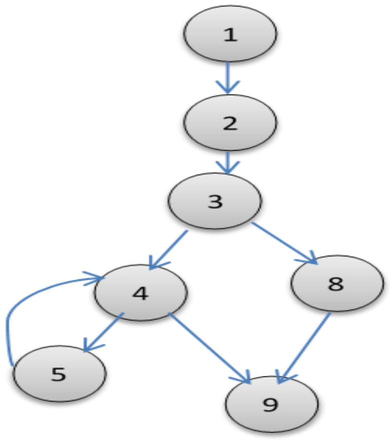
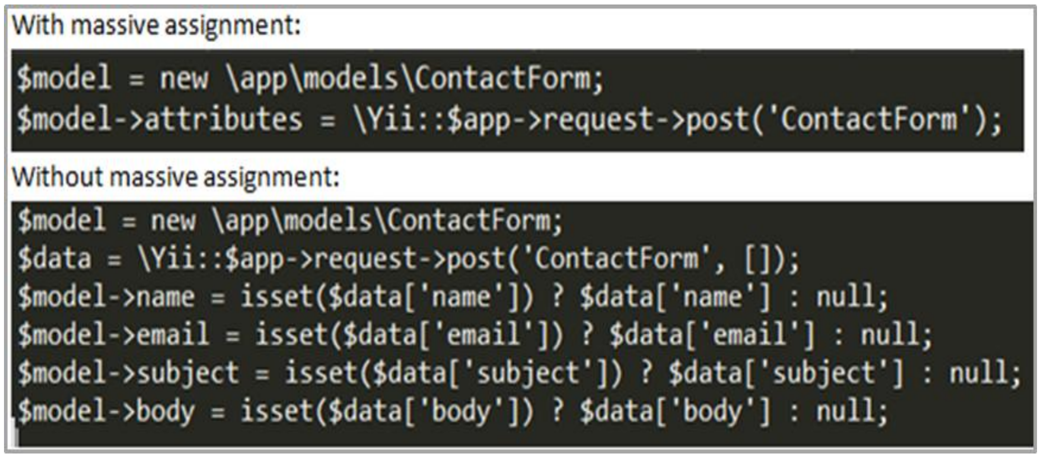

$$\begin{tikzcd}
 & & 1 \arrow[d] & \\
 & & 2 \arrow[d] & \\
 & & 3 \arrow[dl] \arrow[dr] & \\
 & 4 \arrow[dl] \arrow[dr] & & 8 \arrow[dl] \\
5 \arrow[ur, bend left] & & 9 &
\end{tikzcd}$$

FIGURE 2: The flow graph of ordinary PHP code in the previous figure is used to calculate code complexity.

Massive Assignment: The MVC separation mechanism helps the idea of a 'massive assignment'. For example, massive assignment can populate a model with all user's inputs by just one massive assignment. This approach will make the code cleaner and less error-prone compared with the traditional way. For more clarification, the two pieces of code in figure 3 do the same thing; they both assign the inputs of a form submitted by end-users to the attributes of the 'ContactForm' model (the built-in model in Yii framework). Obviously, the code with massive assignments is easier and shorter.


With massive assignment:

```
$model = new \app\models\ContactForm;
$model->attributes = \Yii::$app->request->post('ContactForm');
```

Without massive assignment:

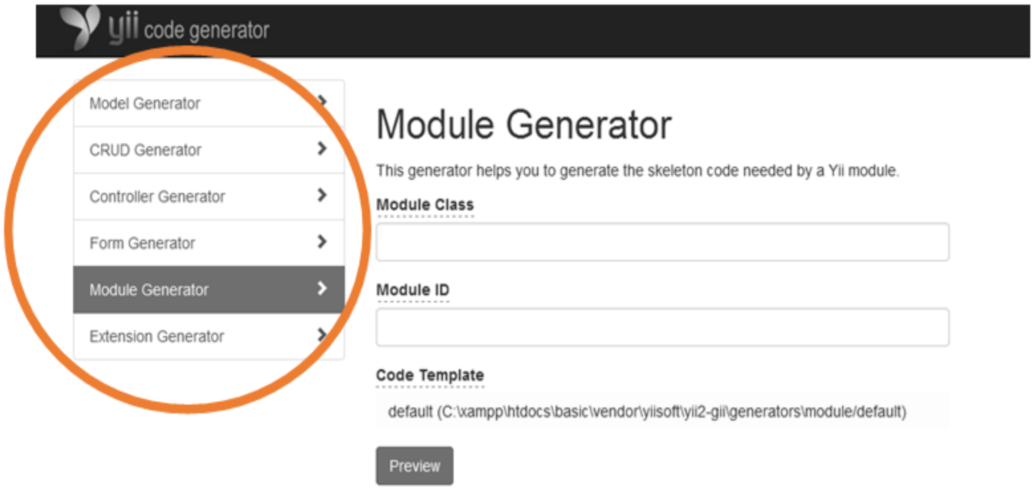
```
$model = new \app\models\ContactForm;
$data = \Yii::$app->request->post('ContactForm', []);
$model->name = isset($data['name']) ? $data['name'] : null;
$model->email = isset($data['email']) ? $data['email'] : null;
$model->subject = isset($data['subject']) ? $data['subject'] : null;
$model->body = isset($data['body']) ? $data['body'] : null;
```

FIGURE 3: Comparing the same codes with and without the massive assignment.

Standardization and Identification: The MVC architecture provides clear standardization and identification of its main components (Model-View-Controller), which helps in finding the parts that have problems. This standardization and identification of MVC parts facilitates the ability to do maintenance, especially replacing components with other components without difficulties. For example, replace the current frontend framework with a different frontend framework in the view part without affecting the logic in the model part.

Automation: The main basic codes in MVC frameworks are usually generated automatically. This contributes to reducing errors. For example, in figure 4 we show how the Yii framework provides several generators for models, controllers, modules, and so on.

Iterative development: The separation of concerns that characterizes the MVC architecture is vital in doing iterative development and refactoring, which supports the maintenance process. Authors in (X. Zhang & Zhang, 2016) provide a study on using MVC and an agile iterative approach. More information about the agile iterative approach and modifiability – reusability attributes can be found in the section 'MVC and Modifiability – Reusability'.



This generator helps you to generate the skeleton code needed by a Yii module.

default (C:\xampp\htdocs\basic\vendor\yiisoft\yii2-gii\generators\module/default)

FIGURE 4: Automatic generators in MVC frameworks (Yii framework in this figure).

Integration and verification: Integration test verifies that the components and modules connected successfully. It provides end-to-end test for the behavior of the system. Many popular MVC frameworks provide facilitations to do integration test.

MVC and the maintenance types: mainly, there are three types of maintenance: corrective maintenance to fix and correct system failures; adaptive maintenance to adapt to new requirements; and perfective maintenance to improve performance and efficiency (Alkhatib, 1992). Regarding corrective maintenance, as we will see in the testability section, we can use debugger modules and components since MVC supports modularity and components. For example, in Yii, we have the debugger module and error-handler component to help corrective maintenance. For adaptive maintenance, as we will see in the extensibility section, MVC supports extensibility to extend the software and append to it new changes to make it up to date. For perfective maintenance, MVC frameworks such as Yii take attention to some performance issues such as lazy and eager loading and offer techniques to enhance the performance. For example, rather than using (1 + N) queries in lazy loading, the Yii framework offers eager loading to have just 2 queries in the ordinary relationship.

The next section provides more details on how MVC supports several maintenance quality attributes.

4. MMERFT MAINTAINABILITY SET

In this section, we introduce the MMERFT maintainability set. As we mentioned, we took benefit from the ISO standard's maintainability attributes that are mentioned in (Bass et al., 2013) to constitute the MMERFT maintainability set. As figure 5 shows, the MMERFT maintainability set consists of Modifiability, Modularity, Extensibility, Reusability, Flexibility, and Testability. The next subsections elaborate on how MVC supports each one of these attributes.

4.1 MVC and Modifiability - Reusability

With its separation mechanism, the MVC architecture facilitates the required modifying and reusing in each part without affecting the other. Thus, MVC facilitates modifying the software to meet user requirements. For instance, frontend developers will be able to make their changes without affecting backend developers' work and vice versa. Modifiability relates to reusability, since the developer often needs to modify the old technologies by reusing new ones to meet user requirements. As we indicated, the MVC architecture helps frameworks to be component-based and to reuse components, which facilitates modifiability and reusability. Another point that affirms the reusability of MVC is that it supports DRY (Don't Repeat Yourself) development (WAGHMARE & ADKAR, 2019). Additionally, MVC gets along with the Agile development style. Authors in (Jegarkandy & Ramsin, 2016) mentioned "Overall, based on the evaluation results, the MVC architectural pattern has been deemed as a suitable pattern for use in agile software development". The Agile style of continuous refinements maximizes reuse(Qian, Fu, Tao, & Xu, 2010), and it facilitates the maintenance as the authors in (Prochazka, 2011) confirmed “the usage of agile principles and techniques also brings a lot of benefits to the support and maintenance”.

Modifiability
Modularity
Testability
MMERFT maintainability set
Extensibility
Flexibility
Reusability

FIGURE 5: MMERFT maintainability set.

MVC in each of its parts simplifies reusing ready-made components, which provide an abstraction that enables developers to reuse and customize the components without the need to look into the details inside them. This feature will minimize the need for writing additional codes, which leads to reducing the probability of mistakes. For example, in the Yii framework regarding the model part, we can use the ‘ActiveRecord’, which is the main database model component, and reuse the ‘ActiveForm’ in the view part to build web forms. By reusing the ‘ActiveRecord’, we can simplify various CRUD manipulations and reduce compatibility issues with various types of databases. The following details give more explanation of the MVC's support for modifiability and reusability in each part.

In Model part: Based on MVC, which assigns a specific part for the data-related logic (Model) and another part for visualizing data (View), that helps make the reuse process based on each part. For example, in the model part, validation rules for the attributes (such as required, string, integer, max and so on) are something that can be reused rather than inventing it each time. Furthermore, these validation rules can be customized and modified based on the requirements. For instance, in the Yii MVC framework, customizing the error message of any rule is available by specifying the message property when declaring the rule as follows:

```
public function rules()
{    return [
        ['username', 'required', 'message' => 'Please choose a username.'],
   ];}
```

In view part: Since MVC separates the business logic of data from the visualization of data, the designer will be able to modify and change the current frontend framework such as the Blueprint CSS framework and reuse other different front-end frameworks such as Bootstrap without any effect on the logic in the model part. Also, modifying and reusing a new theme to rejuvenate the application by using the MVC mechanism becomes very easy. For example, in the Yii framework, it just requires reusing the new layout by putting it in its folder under the view part without any effect on other parts. The same thing goes with reusing and customizing 'Widgets', which is a very important aspect of frontend designing. Widgets are reusable building blocks used in views to create complex and configurable user interface elements in an object-oriented fashion. 'Gridview' is an example of one of the famous widgets in different web frameworks which enables developers to build very interactive tables with pagination, sorting, and so on with just a few lines of code.

In controller part: The controller part in MVC frameworks is usually composed of actions. One of the concepts that shows how MVC architecture facilitates reusability in the controller part is 'Standalone' actions. Standalone actions are usually created to be reused in different controllers or to be redistributed as extensions.

Hence, due to MVC, it has become easier to use famous patterns and components based on each part with a few lines and effective results, such as (ActiveRecord, ActiveForm, Gridview, DataProvider, User Model) which are essential for almost every web application. The straight consequence of reusing will be fewer lines of code, thus fewer errors and bugs. In their research (Cobaleda, Mazo, Becerra, & Duitama, 2016) the authors explained how they used the MVC architecture to provide a reference software architecture for improving the modifiability of personalized web applications. Additionally, in (Otero, 2012) authors confirmed the modifiability and reusability of the MVC architecture and mentioned that MVC is "easy to exchange, enhance and add additional user interfaces".

4.2 MVC and Modularity

Since modules are supposed to have high cohesion and low coupling, modularity is an essential factor in supporting maintainability (Al Dallal, 2009). High cohesion is one of the important metrics in maintainability (Hayes et al., 2004). In many frameworks, MVC is used to build the modules, thus the modules themselves are composites of models, views, controllers, and possibly other components (Safronov & Winesett, 2014). Hence, MVC can help to build the modules based on its architecture, which contributes to supporting modularity and enhancing maintainability. The role of MVC in supporting modularity has been touched upon by authors in (Nitze & Schmietendorf, 2014) and (Grove & Ozkan, 2011).

4.3 MVC and Extensibility

Extensions facilitate adapting the applications to new requirements, which is an important aspect of maintenance (Shekhar, 1998). Extensions can be built as a module or group of modules that can take advantage of the MVC architecture. For example, in the Yii framework, any developer can build an extension based on the MVC module structure, then put it publicly and others will be able to download it and append it to their applications without any effect on the rest of the application. On the other hand, since every part of MVC is clear, based on the separation that MVC offers, it is easier to append a new feature and put every piece of code in its place, which facilitates the extensibility. For instance, it is extremely easy based on MVC to append the ‘Login mechanism’ by email to an application by putting every piece of code in its place and taking full advantage of the email login service. Another example is appending to the view part an extension that just converts the frontend direction of the application from left-right to right-left without any affection to the other parts. Many studies agree on the extensibility of MVC, such as (Henttonen, Matinlassi, Niemelä, & Kanstrén, 2007), (Ibrahim, 2009) and (Wang, Guo, & Song, 2009).

4.4 MVC and Flexibility

As we discussed, MVC supports modifiability, modularity, extensibility, and reusability, which leads to more flexibility and high adaptability to new future requirements that the systems may need. MVC is loosely coupled, and it decreases coupling between components by the separation of three distinct areas of concern (Freeman & Sanderson, 2011), which results in more flexibility. To present how MVC supports flexibility, let us take the ‘scenarios technique’ as an example. Due to the MVC separation mechanism, in the Yii framework a model can be used in different scenarios. For example, the ‘User model’ could be used in the sign-in process to gather login inputs as a scenario, or it could also be used in signing up for the user registration proposal as another scenario. In different scenarios, a model may use different business rules and logic. A specific attribute might be required in one scenario but not in the other scenario. Therefore, using one model with different scenarios gives the developers more flexibility. Additionally, one model could be used by multiple views and themes, which also shows how MVC helps with flexibility. ‘Before and after handling and filters for actions’ another embodiment of how MVC supports the flexibility and many options regarding the actions in the control part. Authors in (Cui, Huang, Liang, & Li, 2009) and (Deinum et al., 2012) confirmed the flexibility of the MVC that it is used to build their proposal as well as the early research published in (Gamma et al., 1993).

4.5 MVC and Testability

The separation logic in MVC leads to a facilitated testing process. As well as in (Yaganteeswarudu & VishnuVardhan, 2017), authors indicated that “The MVC framework provides the applications with testability option. Mainly MVC provides unit testable code “. As we explained, MVC supports modularity and components, which make codes more prepared to be tested based on the ‘unit testing’ approach. Additionally, MVC helps the Test Driven Development (TDD) approach (Zimmerer, 2006), which highly supports testability. The Yii MVC framework, for example, offers unit, function, and acceptance testing and provides ready-to-use test sets for all three test types. Also, it provides modules and components to facilitate testing and detecting errors such as ‘Error Handler’ and ‘Yii Debugger module’ which record and display a lot of debugging information, such as log messages, response statuses, the database queries run, and so on. In (Sanderson, 2009) the authors assured the testability of MVC by stating, "Testability MVC architecture gives you a great start in making your application maintainable and testable, because you will naturally separate different application concerns into different, independent software pieces".

5. DISCUSSION

This study has provided illumination and explanation on many aspects and attributes of MVC’s maintainability, mainly about: how MVC helps reverse engineering in the reengineering process, and reduces lines of code, complexity of code, and the time needed to detect errors. Also, this study highlighted how MVC helps to provide a high level of abstraction and facilitates use of components, parallel maintenance, standardization and identification, automation, and iterative development. Additionally, we pointed out how MVC serves several types of maintenance, such as corrective maintenance, adaptive maintenance, and perfective maintenance. Besides that, we showed by architectural analytic of the MVC’s components mechanism, illustrations of several maintainability attributes, mainly the MMERFT set. The obvious interpretation of all the above outcomes is that MVC has high maintainability and supports the several MMERFT maintainability quality attributes. Since we studied the Yii framework as a case study that represents other frameworks are built based upon MVC, we can generalize these outcomes and explanations to other similar frameworks with an MVC architecture. Developers and researchers that are interested in a particular maintenance aspect or attribute in the MVC architecture will likely find in this study related explanations, details, and references. While most other studies touched on MVC's maintainability among other features or among other architectures, this study is dedicated to detailing and explaining the MVC architecture cross maintenance aspects and attributes particularly.

6. RELATED WORK

By reviewing current studies related to the paper topic, we found that the existing efforts proposed frameworks or patterns based on MVC and stated the resulting features, including maintainability and other attributes. While this research is designed to study the MVC architecture with a focus on maintainability attributes and aspects specifically, the following studies are examples of research that touched MVC and maintainability among other features or among other architectures. For example, both (Kazman et al., 2020) and (Rahmati & Tanhaei, 2021) provided investigations into maintainability issues regarding several architectures and patterns, including MVC, as a part of their research. In the study (Morales-Chaparro et al., 2007), the authors proposed multiple design patterns based on MVC that could be applied to Rich Internet Applications (RIAs). The authors indicated that MVC improves scalability and maintenance. Eventually, they ended up with an 'RIA deeper MVC' design that increases reusability and decreases maintenance time. In their research (Barrett & Delany, 2004), the authors proposed the 'openMVC framework' based on the MVC architecture. They clarified that their suggested framework is designed in a way that allows several of the basic components in the framework to be modified with no code change or recompile requirement. Accordingly, they confirmed that the redundancy is reduced in their framework. They indicated that their framework improved maintenance" The MVC's Controller object role also consists of two components. The Controller Logic component is independent of the View object role and makes the controller workflow logic more readable and easier to maintain as no layout or style information is interspersed within it". In (Gupta & Govil, 2010), the authors established a framework in the J2EE platform and extended it with XML based on the MVC pattern and multi-tier system. Their multi-tier system includes a presentation layer, a business layer, a data persistence layer, and a database layer to separate codes. They pointed out that with this suggested design, their framework becomes more flexible, expansible, reusable and easy to maintain.

However, none of the prior research that we are aware of are devoted to providing an explanatory study on the mechanism of the MVC design primarily from the viewpoint of maintenance and its quality attributes, as this study does. As we saw above, most current studies built frameworks based on MVC and indicated their features, including maintainability. None of them aimed to study maintainability and its attribute with MVC particularly as this study did. Besides that, this study discussed the MVC architecture with several specific issues closely related to maintenance which have not been examined in the same way in current studies. Furthermore, this paper selected real case to explain the ideas and concepts that can be repeated and generalized with other MVC-based frameworks. Additionally, the study gave concrete examples and perceptions that can be applied to improve maintainability and reduce its cost practically.

7. CONCLUSION

This study focused on MVC architecture from the viewpoint of maintenance. It aimed to answer an explanatory question about how MVC architecture supports the maintainability quality attributes. It worked on revealing how MVC-based systems can take advantage of the MVC's high maintainability. We elaborated how MVC supports maintainability in general and in MMERFT quality attributes particularly. Additionally, we examined other maintainability aspects through examples, measures, and graphics such as: complexity of code by using a cyclomatic approach, the re-engineering process, use of components, abstraction, time needed to detect bugs, number of code lines and others. In addition to reviewing various other studies and MVC-based frameworks, the Yii framework was selected as a case study to help fulfill the study's objectives. Developers, maintainers, and project managers can have several practical benefits that can lead to extending software's life and evolving its potential. The study helps to highlight various aspects of architecture that could influence maintenance to be taken in consideration during software development. It will help to revise the maintenance decisions and reduce costs. Researchers and developers seeking more knowledge about a certain maintenance feature or attribute of the MVC architecture will likely find related explanations and references in this study. All of that constituted a clear view on how to utilize MVC's maintainability. The overall conclusion that is derived after conducting this research is that MVC generally supports maintainability and its attributes. Therefore, MVC is a recommended choice in applications that make maintenance a priority. Furthermore, this study leads to other future interesting investigations and questions, such as to which extent the MVC architecture will be able to save its maintainability capabilities in large systems. Also, exploring MVC with other quality attributes such as availability, interoperability, reliability and so on is suggested to be conducted. Another future investigation is studying maintainability with other related Model-View- (MV) patterns such as Model-View-View Model (MVVM) and Model-View-Presenter (MVP).

8. REFERENCES

Akbar, M., & Handriani, I. (2018). *Study and Implementation Information System of Zakat using MVC Architecture.* Paper presented at the IOP Conference Series: Materials Science and Engineering.

Al Dallal, J. (2009). Software similarity-based functional cohesion metric. *IET software, 3*(1), 46-57.

Algestam, H., Offesson, M., & Lundberg, L. (2002). *Using components to increase maintainability in a large telecommunication system.* Paper presented at the Ninth Asia-Pacific Software Engineering Conference, 2002.

Alkhatib, G. (1992). The maintenance problem of application software: An empirical analysis. *Journal of Software Maintenance: Research and Practice, 4*(2), 83-104.

Barrett, R., & Delany, S. J. (2004). *OpenMVC: a non-proprietry component-based framework for web applications.* Paper presented at the Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters.

Bass, L., Clements, P., & Kazman, R. (2013). Software Architecture in Practice. In: Pearson.

Chen, J.-C., & Huang, S.-J. (2009). An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software, 82*(6), 981-992.

Cobaleda, L.-V., Mazo, R., Becerra, J. L. R., & Duitama, J.-F. (2016). Reference software architecture for improving modifiability of personalised web applications-a controlled experiment. *International Journal of Web Engineering and Technology, 11*(4), 351-370.

Cui, W., Huang, L., Liang, L., & Li, J. (2009). *The research of PHP development framework based on MVC pattern.* Paper presented at the 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology.

Deinum, M., Serneels, K., Yates, C., Ladd, S., Vervaet, E., & Vanfleteren, C. (2012). *Pro Spring MVC: With Web Flow*: Apress.

Feiler, P. H. (1993). *Reengineering: An engineering problem*.

Freeman, A., & Sanderson, S. (2011). The MVC Pattern. In *Pro ASP. NET MVC 3 Framework* (pp. 63-88): Springer.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). *Design patterns: Abstraction and reuse of object-oriented design.* Paper presented at the European Conference on Object-Oriented Programming.

Gining, R. A. J., Fauzi, S. S. M., Jamaluddin, N. F., Azmi, N. A. M., Hashim, M. A., Fuzi, M. F. M., & Ibrahim, A. F. (2018). Adaptation of. NET MVC Framework in Developing an Agriculture Sources Inventory System. *Journal of Computing Research and Innovation, 3*(3), 6-13.

Grove, R. F., & Ozkan, E. (2011). *The MVC-web Design Pattern.* Paper presented at the WEBIST.

Gupta, P., & Govil, M. C. (2010). MVC Design Pattern for the multi framework distributed applications using XML, spring and struts framework. *International Journal on Computer Science and Engineering, 2*(04), 1047-1051.

Hayes, J. H., Patel, S. C., & Zhao, L. (2004). *A metrics-based software maintenance effort model.* Paper presented at the Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings.

Henttonen, K., Matinlassi, M., Niemelä, E., & Kanstrén, T. (2007). Integrability and extensibility evaluation from software architectural models–A case study. *The Open Software Engineering Journal, 1*(1).

Ibrahim, E. (2009). *ASP. NET MVC 1.0 Test Driven Development: Problem-Design-Solution*: Wrox Press Ltd.

Jegarkandy, S. S., & Ramsin, R. (2016). Assessing the suitability of architectural patterns for use in agile software development. In: PATTERNS.

Kaur, A., Kaur, P., & Kaushal, P. (2020). Maintainability Procedure in Component-Based Software. *Journal of Computational and Theoretical Nanoscience, 17*(11), 5156-5161.

Kazman, R., Bianco, P., Ivers, J., & Klein, J. (2020). *Maintainability*.

Kumar, A., Kumar, R., & Grover, P. (2007). An evaluation of maintainability of aspect-oriented systems: a practical approach. *International Journal of Computer Science and Security, 1*(2), 1-9.

Majeed, A., & Rauf, I. (2018). MVC architecture: a detailed insight to the modern web applications development. *Peer Review Journal of Solar & Photoenergy Systems, 1*(1), 1-7.

Molnar, A.-J., & Motogna, S. (2020). A Study of Maintainability in Evolving Open-Source Software. *arXiv preprint arXiv:2009.00959*.

Morales-Chaparro, R., Linaje, M., Preciado, J., & Sánchez-Figueroa, F. (2007). MVC web design patterns and rich internet applications. *Proceedings of the Jornadas de Ingenierıa del Software y Bases de Datos*, 39-46.

Nitze, A., & Schmietendorf, A. (2014). *Modularity of javascript libraries and frameworks in modern web applications.* Paper presented at the Selected Topics to the User Conference on Software Quality, Test and Innovation.

Otero, C. (2012). *Software engineering design: theory and practice*: CRC Press.

Prochazka, J. (2011). Agile support and Maintenance of IT services. In *Information Systems Development* (pp. 597-609): Springer.

Qian, K., Fu, X., Tao, L., & Xu, C.-w. (2010). *Software architecture and design illuminated*: Jones & Bartlett Learning.

Rahmati, Z., & Tanhaei, M. (2021). Ensuring software maintainability at software architecture level using architectural patterns. *AUT Journal of Mathematics and Computing, 2*(1), 81-102.

Rosenberg, L. H., & Hyatt, L. E. (1996). Software re-engineering. *Software Assurance Technology Center*, 2-3.

Safronov, M., & Winesett, J. (2014). *Web application development with Yii 2 and PHP*: Packt Publishing Ltd.

Sanderson, S. (2009). *ASP. Net MVC Framework Preview*: Apress.

Seth, K., Sharma, A., & Seth, A. (2009). Component selection efforts estimation–a fuzzy logic based approach. *International Journal of Computer Science and Security,(IJCSS), 3*(3), 210-215.

Sharma, A., Kumar, R., & Grover, P. (2007). Managing component-based systems with reusable components. *International Journal of Computer Science and Security, 1*(2), 52-57.

Shekhar, S. (1998). Perfective Maintenance & Extensibility. Retrieved from http://www-users.cs.umn.edu/~shekhar/5180/notes/ch13.html

WAGHMARE, D. V., & ADKAR, P. (2019). Agile Development using Ruby on Rails Framework.

Wang, Y., Guo, C., & Song, L. (2009). *Architecture of E-Commerce Systems Based on J2EE and MVC Pattern.* Paper presented at the 2009 International Conference on Management of e-Commerce and e-Government.

Wieringa, R. J. (2014). *Design science methodology for information systems and software engineering*: Springer.

Winesett, J. (2012). *Web Application development with Yii and PHP*: Packt Publishing Ltd.

Xu, B., & Liao, Y. (2021). *The Online Exam System Research Based on the MVC Framework.* Paper presented at the 2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC).

Yaganteeswarudu, A., & VishnuVardhan, Y. (2017). *Software appication to prevent suicides of farmers with asp. net MVC.* Paper presented at the 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence.

Zhang, H. (2009). *An investigation of the relationships between lines of code and defects.* Paper presented at the 2009 IEEE International Conference on Software Maintenance.

Zhang, X., & Zhang, Z. (2016). *Research on Web Development Paradigm based on MVC Pattern and Agile Development Mode.* Paper presented at the 2016 5th International Conference on Advanced Materials and Computer Science (ICAMCS 2016).

Zimmerer, P. (2006). *Insights in Real Test-Driven Development*. Paper presented at the Conference of the Association for Software Testing (CAST), Indianapolis, Indiana, USA.